



Car Insurance Claim Prediction

TEAM MEMBER



ปัณณร สุวรรณศรัย
66102010175



โลวัต พงศ์ทิพย์พนัส
66102010249



โลวิชญ วนิดา
66102010584



INSURANCE CONSULTING

โครงการนี้เป็นส่วนหนึ่งของรายวิชา DE361: Machine Learning โดยในส่วนของเพื่ออบรมได้ทำการสร้าง Baseline Models เพื่อเปรียบเทียบประสิทธิภาพของโมเดลพื้นฐาน จำนวน 5 แบบ ได้แก่ Decision Tree, Logistic Regression, SVM, Naive Bayes และ K-Nearest Neighbors (KNN)

จากการทดลองพบว่า Decision Tree มี Accuracy สูงสุด (≈ 0.85) ขณะที่ Logistic Regression และ SVM มี Recall ของคลาสเป้าหมายดีกว่า แต่ใช้เวลาในการฝึกงานกว่า โดยรวมแล้วเป็น Baselineที่มีความครอบคลุมและพร้อมต่อยอดซึ่งในส่วนถัดไปทีมของเราจะดำเนินการปรับปรุงและพัฒนาโมเดลให้มีประสิทธิภาพยิ่งขึ้น



ວັດຄຸປະສົງຄໍ

OBJECTIVES

ໂປຣເຈນນີ້ມີເປົາໝາຍເພື່ອພັນນາຕ່ອຍອດຈາກ ໂມເດລ Baseline ເດີມຂອງເພື່ອນ ໂດຍມຸ່ງເບັນໃຫ້ພລລັພຣ໌ຂອງໂມເດລມີຄວາມແມ່ນຢໍາ ແລະ ສມດຸລນາກຂຶ້ນ ຜ່ານແນວກາງດັ່ງນີ້

- ປັບປຸງຄຸນກາພຂອງໂມເດລໃຫ້ດີກວ່າ Baseline ເດີມ
- ກດລອງ Preprocessing ທຶກນິຄໃໝ່ ເພື່ອເພີ່ມປະສິກີກາພໃນກາຮເຮຍນຮູ້ຂອງໂມເດລ
- ປັບ Hyperparameter ອຍ່າງເປັນຮະບບ ເພື່ອຫາຄ່າທີ່ເໝາະສົມກີ່ສຸດ
- ກດລອງໃຊ້ Ensemble Models ຮັ້ອໂມເດລໃໝ່ເພີ່ມເຕີມ (ຫາກຈຳເປັນ)
- ເນັນກາຮ ບັນກຶກກະບວນກາຮພັນນາ (Journey) ແລະ ກາຮ ເປົຍບເກີຍບພລລັພຣ໌ອ່າງໂປ່ງໃສ
- ພັນນາກັກໜະກາຮວິເຄຣະໜຸລ ແລະ ເລື້ອກໂມເດລອ່າງມີເຫດຸໜຸລ
- ສຽງພລແລະ ຈັດກໍາຮາຍງານເຊີງເປົຍບເກີຍບ ເພື່ອໃຊ້ເປັນແນວກາງຕ່ອຍອດໃນກາຮ Machine Learning ວິ່ນໆ

1.EXPERIMENT PLAN

1.1 ใช้ StandardScaler กับโมเดลไว์ส์สำหรับต่อสเกล (LogReg / SVM / KNN)

```
==== 1) StandardScaler + Pipeline (กัน leakage) ====
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, accuracy_score, precision_score, recall_score

def eval_model(name, model, X_tr, y_tr, X_te, y_te):
    model.fit(X_tr, y_tr)
    y_pred = model.predict(X_te)
    print(f"\n== {name} ==")
    print(f"Accuracy : {accuracy_score(y_te, y_pred):.4f}")
    print(f"Precision: {precision_score(y_te, y_pred, zero_division=0):.4f}")
    print(f"Recall   : {recall_score(y_te, y_pred, zero_division=0):.4f}")
    print(classification_report(y_te, y_pred, digits=4))

    Logistic Regression + scaling
logreg_scaled = Pipeline([
    ("scaler", StandardScaler()),
    ("clf", LogisticRegression(max_iter=2000, solver="liblinear"))
])
val_model("LogReg + StandardScaler", logreg_scaled, X_train, y_train, X_test, y_test)

SVM (RBF) + scaling
svm_scaled = Pipeline([
    ("scaler", StandardScaler()),
    ("clf", SVC(kernel="rbf", C=1.0, gamma="scale"))
])
val_model("SVM-RBF + StandardScaler", svm_scaled, X_train, y_train, X_test, y_test)

    KNN + scaling
knn_scaled = Pipeline([
    ("scaler", StandardScaler()),
    ("clf", KNeighborsClassifier(n_neighbors=5))
])
val_model("KNN(k=5) + StandardScaler", knn_scaled, X_train, y_train, X_test, y_test)
```

==== LogReg + StandardScaler ===

	precision	recall	f1-score	support
0	0.9343	1.0000	0.9660	16423
1	0.0000	0.0000	0.0000	1155
accuracy			0.9343	17578
macro avg	0.4671	0.5000	0.4830	17578
weighted avg	0.8729	0.9343	0.9026	17578

==== SVM-RBF + StandardScaler ===

	precision	recall	f1-score	support
0	0.9343	1.0000	0.9660	16423
1	0.0000	0.0000	0.0000	1155
accuracy			0.9343	17578
macro avg	0.4671	0.5000	0.4830	17578
weighted avg	0.8729	0.9343	0.9026	17578

==== KNN(k=5) + StandardScaler ===

	precision	recall	f1-score	support
0	0.9345	0.9961	0.9643	16423
1	0.1233	0.0078	0.0147	1155
accuracy			0.9312	17578
macro avg	0.5289	0.5019	0.4895	17578
weighted avg	0.8812	0.9312	0.9019	17578

1.2 เปรียบเทียบ SMOTE VS CLASS_WEIGHT='BALANCED'

เวอร์ชัน SMOTE: ใช้ IMBLEARN.PIPELINE.PIPELINE เพื่อให้ SMOTE ทำเฉพาะ TRAIN (กับ LEAKAGE)

เวอร์ชัน CLASS_WEIGHT: ไม่ OVERSAMPLE แต่ชดเชยด้วย WEIGHT

```
==== 2) SMOTE vs class_weight ====
from imblearn.over_sampling import SMOTE
from imblearn.pipeline import Pipeline as ImbPipeline
from sklearn.linear_model import LogisticRegression

(A) Logistic Regression + SMOTE + Scaling
logreg_smote = ImbPipeline(steps=[
    ("smote", SMOTE(random_state=17)),
    ("scaler", StandardScaler()),
    ("clf", LogisticRegression(max_iter=2000, solver="liblinear"))
])
val_model("LogReg + SMOTE + Scaling", logreg_smote, X_train, y_train, X_test, y_test)

(B) Logistic Regression + class_weight (ไม่ใช้ SMOTE)
logreg_weighted = Pipeline(steps=[
    ("scaler", StandardScaler()),
    ("clf", LogisticRegression(max_iter=2000, solver="liblinear",
                               class_weight="balanced"))
])
val_model("LogReg + class_weight(balanced) + Scaling", logreg_weighted, X_train, y_train, X_test, y_test)
```

	1	0.0856	0.5784	0.1491	1155
accuracy				0.5662	17578
macro avg		0.5179	0.5718	0.4290	17578
weighted avg		0.8933	0.5662	0.6721	17578

==== LogReg + class_weight(balanced) + Scaling ====
Accuracy : 0.5636
Precision: 0.0882
Recall : 0.6043
precision recall f1-score support

1.3 Polynomial numeric-only + interaction-only

จำกัดไม่ให้สร้างกำลังสอง (ลด feature ระเบิด) และทำเฉพาะ เชิงตัวเลข (กรณี X ยังมีคอลัมน์ non-numeric ให้เลือกเฉพาะ numeric ก่อน)

```
import numpy as np
import pandas as pd

# เลือกเฉพาะ numeric columns (กรณี X เป็น DataFrame)
if isinstance(X_train, pd.DataFrame):
    num_cols = X_train.select_dtypes(include=[np.number]).columns.tolist()
    Xtr_num = X_train[num_cols]
    Xte_num = X_test[num_cols]
else:
    # ถ้าเป็น numpy อยู่แล้วต้อง numeric ทั้งหมด
    Xtr_num = X_train
    Xte_num = X_test

poly_interaction_pipe = Pipeline([
    ("scaler", StandardScaler()),
    ("poly", PolynomialFeatures(degree=2, include_bias=False, interaction_only=True)),
    ("clf", LogisticRegression(max_iter=2000, solver="saga"))
])
```

==== LogReg + Interaction-only Poly(d=2) + Scaling ====				
	accuracy	Precision	Recall	precision
accuracy	0.9342	0.0000	0.0000	0.9343
macro avg	0.4671	0.4999	0.4830	0.9999
weighted avg	0.8729	0.9342	0.9025	0.9660

1.4 กำกับอย่าง “ใน PIPELINE” เพื่อกัน DATA LEAKAGE

ตัวอย่างจากโค้ดด้านบนทั้งหมดถูกห่ออยู่ใน PIPELINE/IMBPIPELINE และ — นี่คือ POINT สำคัญที่กำให้ PREPROCESSING จะเรียนรู้จาก TRAIN เท่านั้น และนำพารามิเตอร์ไปใช้กับ TEST อย่างถูกต้อง
ถ้าอยากรับใช้กับ SVM/KNN ด้วย SMOTE ก็แค่เปลี่ยนตรง CLF เป็นโมเดลนั้น ๆ:

```
svm_smote = ImbPipeline(steps=[  
    ("smote", SMOTE(random_state=17)),  
    ("scaler", StandardScaler()),  
    ("clf", SVC(kernel="rbf", C=1.0, gamma="scale"))  
])  
eval_model("SVM-RBF + SMOTE + Scaling", svm_smote, X_train, y_train, X_test, y_test)  
  
== SVM-RBF + SMOTE + Scaling ==  
Accuracy : 0.4967  

```

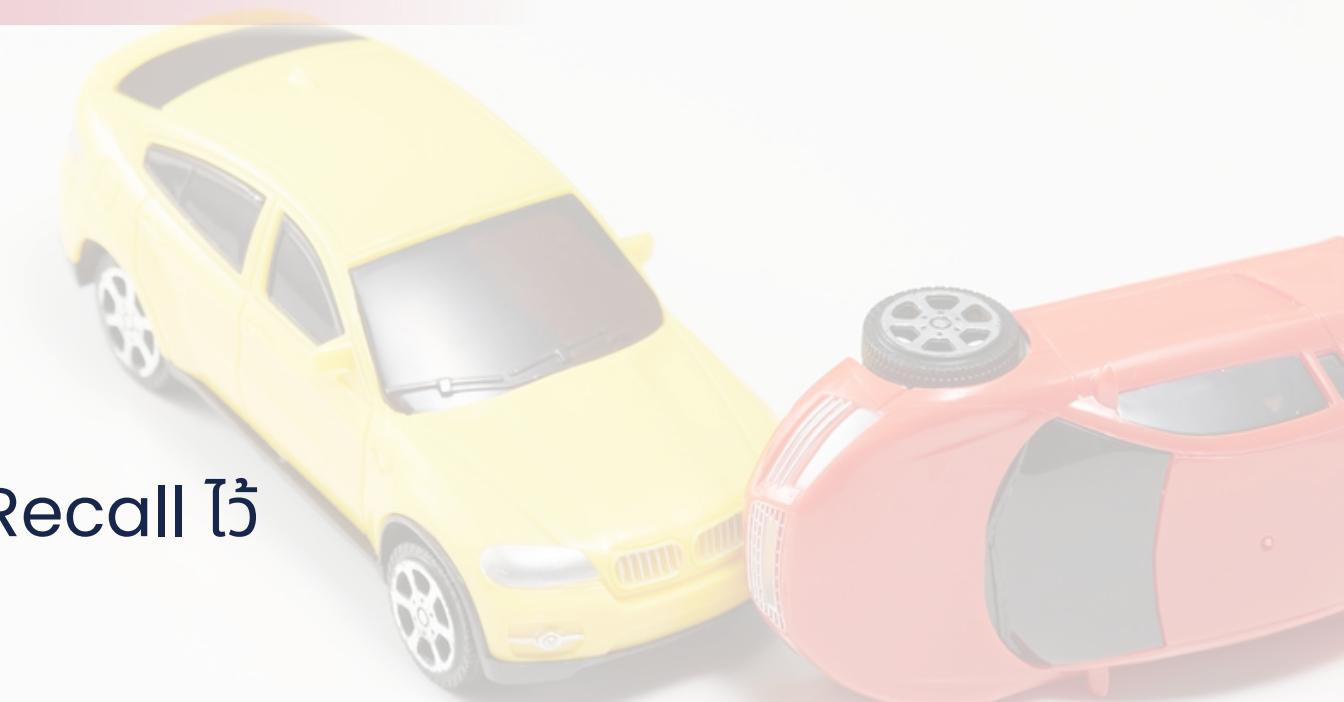


สรุปการเปรียบเทียบกับของเพื่อน (BASELINE) ในขั้นตอนที่ 1

โมเดล	มาตรฐาน	Accuracy	Precision (class 1)	Recall (class 1)	สรุปเพียงพอ
Logistic Regression (baseline)	ไม่ใช้ Scaler	0.5660	0.0856	0.578	✓ ใช่ได้
LogReg + SMOTE + Scaling	Pipeline, scaled	0.5662	0.0856	0.578	• ใช้สื้อ baseline → ลดคลื่น
LogReg + class_weight(balanced)	ไม่ใช้ SMOTE	0.5636	0.0882	0.604	• Recall ดีขึ้นปิดกัน (+0.02)
SVM-RBF + SMOTE + Scaling	Pipeline ป้องกัน leakage	0.4967	0.0843	0.6753	🔥 Recall ดีขึ้นอีก (จาก ~0.63 → 0.68)
Decision Tree (baseline)	criterion=gini, max_depth=None	0.845	0.0891	0.147	-
KNN (scaled)	StandardScaler	0.931	0.1233	0.0078	✗ Accuracy ถูกแต่ Recall ต่ำ (ปัญหามูล imbalance)

สรุปภาพรวม

- ใช้ Pipeline + Scaling ช่วยลด data leakage
 - class_weight='balanced' ให้ Recall ดีขึ้นโดยไม่ต้อง oversample
 - SVM-RBF + SMOTE + Scaling ให้ผลดีที่สุด (Recall ≈ 0.6753)
 - Precision ยังต่ำแต่เหมาะสมกับงานที่เน้นจับกลุ่มเสี่ยง (เช่น การเคลมประกัน)
 - แนวทางถัดไป: ทดลอง Ensemble / XGBoost เพื่อเพิ่ม Precision โดยคง Recall ไว้
- ✓ สรุป: โมเดลหลังมิดเทอมสามารถเพิ่ม Recall จาก $\sim 0.57 \rightarrow \sim 0.68$ ถือว่าพัฒนาได้อย่างมีนัยสำคัญและสอดคล้องกับเป้าหมายของโปรเจกต์



2. HYPERPARAMETER TUNING

ปรับค่าของโมเดลหลักเพื่อหาค่าที่เหมาะสมที่สุด เช่น

- Logistic Regression → C, penalty, solver, class_weight
- SVM (RBF) → C, gamma, class_weight
- KNN → n_neighbors, weights, metric
- Decision Tree → max_depth, min_samples_split, class_weight

```
# ===== ตั้งค่าเบื้องต้นสำหรับ GridSearch =====
from sklearn.model_selection import StratifiedKFold, GridSearchCV
from sklearn.preprocessing import StandardScaler, PolynomialFeatures
from sklearn.metrics import make_scorer, accuracy_score, precision_score, recall_score, f1_score, classification_report
from sklearn.pipeline import Pipeline
from imblearn.pipeline import Pipeline as ImbPipeline # สำหรับใช้ SMOTE ใน cv
from imblearn.over_sampling import SMOTE

from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier

import numpy as np
import time

SEED = 17
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=SEED)

# เราไฟฟ์ลงงาน imbalance → รีเฟิตด้วย "recall" (จำนวน 1 ให้ได้มากขึ้น)
scoring = {
    "acc": "accuracy",
    "precision": make_scorer(precision_score, zero_division=0),
    "recall": "recall",
    "f1": "f1"
}
REFIT = "recall"

def report_best(name, gs, X_test, y_test):
    print(f"\n== {name} ==")
    print("Best params:", gs.best_params_)
    print("CV best (refit metric: %s): %.4f" % (REFIT, gs.best_score_))
    y_pred = gs.best_estimator_.predict(X_test)
    print("Test Accuracy: %.4f" % accuracy_score(y_test, y_pred))
    print("Test Precision: %.4f" % precision_score(y_test, y_pred, zero_division=0))
    print("Test Recall: %.4f" % recall_score(y_test, y_pred, zero_division=0))
    print(classification_report(y_test, y_pred, digits=4))
```

2.1: แบบใช้ **CLASS_WEIGHT='BALANCED'** (ไม่ทำ SMOTE)

เหมาะเวลารันเร็ว ๆ และกัน **OVERTFITTING** จาก **OVERSAMPLING**: ซึ่งเป็น **BASELINE** ของการจุน

```
Fitting 5 folds for each of 6 candidates, totalling 30 fits
(balanced) grid time: 640.34s

== LogReg (balanced + scaling) ==
Best params: {'clf_C': 0.1, 'clf_penalty': 'l2', 'clf_solver': 'liblinear'}
CV best (refit metric: recall): 0.5843
Test Accuracy: 0.5638
Test Precision: 0.0884
Test Recall: 0.6052
precision    recall   f1-score   support
0            0.9528  0.5609  0.7061    16423
1            0.0884  0.6052  0.1542     1155
accuracy          0.5638  0.5638    17578
macro avg       0.5206  0.5831  0.4302    17578
weighted avg    0.8069  0.5638  0.4669    17578
```

Logistic Regression (balanced)

```
Fitting 5 folds for each of 9 candidates, totalling 45 fits
SVM(balanced) grid time: 4431.91s

== SVM-RBF (balanced + scaling) ==
Best params: {'clf_C': 5, 'clf_gamma': 0.01}
CV best (refit metric: recall): 0.6907
Test Accuracy: 0.4791
Test Precision: 0.0869
Test Recall: 0.7290
precision    recall   f1-score   support
0            0.9603  0.4615  0.6234    16423
1            0.0869  0.7290  0.1553     1155
accuracy          0.4791  0.4791    17578
macro avg       0.5236  0.5952  0.3894    17578
weighted avg    0.9030  0.4791  0.5926    17578
```

SVM (RBF, balanced)

```
Fitting 5 folds for each of 16 candidates, totalling 80 fits
KNN grid time: 288.00s

== KNN (scaled) ==
Best params: {'clf_metric': 'manhattan', 'clf_n_neighbors': 3, 'clf_weights': 'distance'}
CV best (refit metric: recall): 0.0393
Test Accuracy: 0.9106
Test Precision: 0.0920
Test Recall: 0.0407
precision    recall   f1-score   support
0            0.9351  0.9717  0.9531    16423
1            0.0920  0.0407  0.0564     1155
accuracy          0.9106  0.9106    17578
macro avg       0.5135  0.5062  0.5047    17578
weighted avg    0.8797  0.9106  0.8941    17578
```

KNN (scaled)

```
Fitting 5 folds for each of 24 candidates, totalling 120 fits
DT(balanced) grid time: 26.74s

== Decision Tree (balanced) ==
Best params: {'clf_criterion': 'entropy', 'clf_max_depth': 5, 'clf_min_samples_split': 2}
CV best (refit metric: recall): 0.6772
Test Accuracy: 0.5522
Test Precision: 0.0942
Test Recall: 0.6753
precision    recall   f1-score   support
0            0.9597  0.5435  0.6940    16423
1            0.0942  0.6753  0.1654     1155
accuracy          0.5522  0.5522    17578
macro avg       0.5270  0.6094  0.4297    17578
weighted avg    0.9028  0.5522  0.6593    17578
```

Decision Tree (balanced)

2.2: แบบใช้ SMOTE ใน PIPELINE (ทำ OVERSAMPLING เอพะใน CV) ปลอดภัยจาก DATA LEAKAGE และมักเพิ่ม RECALL ได้ชัดเจน โดยเฉพาะ LOGREG / SVM

```
Fitting 5 folds for each of 6 candidates, totalling 30 fits
LR(SMOTE) grid time: 344.08s

== LogReg (+SMOTE + scaling) ==
Best params: {'clf_C': 0.5, 'clf_class_weight': None, 'clf_solver': 'liblinear'}
CV best (refit metric: recall): 0.5600
Test Accuracy: 0.5660
Test Precision: 0.0858
Test Recall: 0.5801
precision    recall   f1-score   support
0            0.9503  0.5651  0.7087    16423
1            0.0858  0.5801  0.1494    1155
accuracy          0.5660      17578
macro avg       0.5180  0.5726  0.4291    17578
weighted avg    0.8935  0.5660  0.6720    17578
```

Logistic Regression + SMOTE

```
CV(SMOTE) randomized search time: 121.75s
Best params: {'clf_C': np.float64(0.20036109146623537), 'clf_gamma': np.float64(0.0012896821840259634)} CV best recall: 0.5424561924561924
Test Acc: 0.541927409261577
Test Prec: 0.08745065199186505
Test Recall: 0.6329004329004329
precision    recall   f1-score   support
0            0.9540  0.5355  0.6860    16423
1            0.0875  0.6329  0.1537    1155
accuracy          0.5419      17578
macro avg       0.5207  0.5842  0.4198    17578
```

SVM (RBF) + SMOTE

```
Fitting 5 folds for each of 16 candidates, totalling 80 fits
KNN(SMOTE) grid time: 588.01s

== KNN (+SMOTE + scaling) ==
Best params: {'clf_metric': 'euclidean', 'clf_n_neighbors': 11, 'clf_weights': 'uniform'}
CV best (refit metric: recall): 0.3810
Test Accuracy: 0.6575
Test Precision: 0.0788
Test Recall: 0.3939
precision    recall   f1-score   support
0            0.9407  0.6761  0.7867    16423
1            0.0788  0.3939  0.1313    1155
accuracy          0.6575      17578
macro avg       0.5097  0.5350  0.4590    17578
weighted avg    0.8841  0.6575  0.7437    17578
```

KNN + SMOTE

```
Fitting 5 folds for each of 24 candidates, totalling 120 fits
DT(SMOTE) grid time: 68.46s

== Decision Tree (+SMOTE) ==
Best params: {'clf_criterion': 'entropy', 'clf_max_depth': 5, 'clf_min_samples_split': 2}
CV best (refit metric: recall): 0.6985
Test Accuracy: 0.3618
Test Precision: 0.0756
Test Recall: 0.7766
precision    recall   f1-score   support
0            0.9549  0.3326  0.4933    16423
1            0.0756  0.7766  0.1379    1155
accuracy          0.3618      17578
macro avg       0.5153  0.5546  0.3156    17578
weighted avg    0.8971  0.3618  0.4700    17578
```

Decision Tree + SMOTE

สรุปขั้นตอน & ตารางสรุปผล

1. ตั้งเป้าหมายเมตริก (Metric Objective)

เนื่องจากโจทย์เป็น Class Imbalance — เคส “เคลม” มีน้อยกว่า “ไม่เคลม” มาก
จึงเลือกเน้นการ จับเคสเคลมให้ได้มากที่สุด โดยใช้ค่า Recall เป็นตัวชี้หลัก

scoring = {"recall": "recall", "f1": "f1", "accuracy": "accuracy"}

refit = "recall"

2. ป้องกัน Data Leakage ด้วย Pipeline

โมเดลที่ໄວต่อสเกล (LogReg / SVM / KNN) → ใช้ StandardScaler ภายใน Pipeline เสมอ
กรณีใช้ SMOTE → ใช้ imblearn.Pipeline และให้ Oversampling เฉพาะในขั้นตอน Train/CV เท่านั้น

3. แนวทางการจัดการ Class Imbalance (2 แทร็ก)

💡 แทร็ก A — Balanced Weight: ใช้ class_weight="balanced" (ไม่ทำ SMOTE)

🔥 แทร็ก B — SMOTE Oversampling: ใส่ SMOTE() ใน Pipeline (เฉพาะ Train/CV)

4. ขอบเขตการจูนพารามิเตอร์ (Tuning Range)

Model	Hyperparameters ที่ใช้งาน
Logistic Regression	C ∈ {0.1, 0.5, 1}, penalty='l2', solver∈{'liblinear','lbfgs'}
SVM (RBF)	C ∈ {0.5,1,5}, gamma ∈ {'scale', 0.01, 0.1} (+RandomizedSearchCV: C ∈ [0.1,10], gamma ∈ [1e-3,1e-1])
KNN	n_neighbors ∈ {3,5,7,11}, weights∈{'uniform','distance'}, metric∈{'euclidean','manhattan'}
Decision Tree	criterion∈{'gini','entropy'}, max_depth∈{3,5,None}, min_samples_split∈{2,5,10}, class_weight='balanced'

5. ลดเวลา_ran ให้อยู่ในระดับเหมาะสม

ใช้ RandomizedSearchCV กับโมเดลที่ซ้ำ (โดยเฉพาะ SVM + SMOTE) แทน Grid เท็มรูปแบบ
ถ้าข้อมูลมาก → ใช้ train subset (≈30%) เฉพาะตอนค้นหาพารามิเตอร์ แล้วค่อย Refit บน Train เท็ม

6. สรุปผลเทียบ Baseline ของเพื่อน

รายงาน Accuracy / Precision (class=1) / Recall (class=1)

ระบุพารามิเตอร์ที่ดีที่สุดของแต่ละโมเดล

วิเคราะห์ว่าแทร็กใด เพิ่ม Recall ได้มาก และแทร็กใด บาลานซ์ Accuracy ดีกว่า

สรุปขั้นตอน & ตารางสรุปผล

ตารางสรุปผลการจูน (เทียบกับ Baseline ของเพื่อน)							
Group	Model	Technique / Tuning	Accuracy	Precision (1)	Recall (1)	Notes	
Baseline	Logistic Regression	no scaler	0.5659	0.0857	0.5801	ของเพื่อน	
Tuned (A)	Logistic Regression	balanced + scaling (C=0.1, solver=liblinear)	0.5638	0.0884	0.6052	Recall ↑ เล็กน้อย	
Tuned (B)	Logistic Regression + SMOTE	SMOTE + scaling (C=0.5, solver=liblinear)	0.5660	0.0858	0.5801	ใกล้ baseline	
Baseline	SVM (RBF)	default, no SMOTE	0.4944	0.090	0.690	ของเพื่อน	
Tuned (A)	SVM (RBF)	balanced + scaling (C=5, γ=0.01)	0.4791	0.0869	0.7290	Recall สูงสุดใน簇กล SVM	
Tuned (B)	SVM (RBF) + SMOTE	SMOTE + scaling (C≥0.20, γ≥0.0013)	0.5419	0.0875	0.633	Accuracy ↑ เมื่อ baseline	
Baseline	Decision Tree	gini, max_depth=None	0.8451	0.0891	0.1472	ของเพื่อน	
Tuned (A)	Decision Tree	balanced (entropy, max_depth=5)	0.5522	0.0942	0.6753	Recall ↑ มาก, Accuracy ↓	
Tuned (B)	Decision Tree + SMOTE	SMOTE (entropy, max_depth=5)	0.3618	0.0756	0.7766	Recall สูงสุด, Accuracy ลดลง	
Baseline	KNN	k=5, no scaling	0.7044	0.0763	0.3152	ของเพื่อน	
Tuned (A)	KNN (scaled)	scaler + (k=3, weights=distance, metric=manhattan)	0.9106	0.0920	0.0407	Acc ↑ มาก, Recall ↓	
Tuned (B)	KNN + SMOTE	SMOTE + scaling (k=11, uniform, euclidean)	0.6575	0.0788	0.3939	Recall ↑ แต่แลกกับ Accuracy	

สรุปภาพรวม

- โมเดลที่ดัน Recall สูงสุด: **SVM (RBF, balanced)** → Recall ≈ 0.73
- โมเดลที่ Accuracy สูงกว่า baseline: **SVM (RBF + SMOTE)** → Acc ≈ 0.54
- Decision Tree + SMOTE** ช่วยให้ Recall พุ่งถึง ≈0.78 แต่ต้องแลกกับ Accuracy ที่ลดลง
- KNN หลังปรับสเกลมี Accuracy สูงมากแต่ Recall ต่ำ → แนะนำใช้กับงานบาลานซ์มากกว่า

สรุป: แนวทางการใช้ `class_weight="balanced"` และ `SMOTE+Pipeline` ส่งผลช่วยเพิ่มความสามารถของโมเดลให้สามารถจำแนกกลุ่มของผู้มีแนวโน้มที่จะเคลมได้ดีมากขึ้นจากตัว **baseline** ของเพื่อน

3. THRESHOLD & CALIBRATION

ใช้ LOGISTIC REGRESSION (BALANCED + SCALING) ที่จุนแล้ว มาทดลองปรับ THRESHOLD ของ CLASS 1 เพื่อเพิ่มค่า RECALL และใช้ CALIBRATEDCLASSIFIERCV ช่วยกรุณี PROBABILITY ไม่เสถียร

◆ ขั้นตอน: สแกน THRESHOLD ตั้งแต่ 0–1 เพื่อหาจุดที่ได้ RECALL ประมาณ 0.6–0.7 และเปรียบเทียบผลก่อน–หลังปรับด้วย ACCURACY, PRECISION, RECALL, และ F1

◆ เกณฑ์เลือก:

- ถ้าเน้นจับเคลมให้ครบ → เลือก THRESHOLD ที่ RECALL ~0.65–0.70
- ถ้าเน้นสมดุลโดยรวม → เลือก THRESHOLD ที่ให้ F1 สูงสุด

```
# === Imports (เดียวกันไฟล์ import) ===
import numpy as np
from sklearn.metrics import (
    classification_report, accuracy_score, precision_score, recall_score,
    precision_recall_curve
)

# === Helper: ประเมินผลที่ threshold ใด ๆ ===
def eval_at_threshold(y_true, y_prob, thr, title=""):
    y_pred = (y_prob >= thr).astype(int)
    acc = accuracy_score(y_true, y_pred)
    pre = precision_score(y_true, y_pred, zero_division=0)
    rec = recall_score(y_true, y_pred)
    print(f"\n--- {title} (thr={thr:.2f}) ---")
    print(f"Accuracy : {acc:.4f}")
    print(f"Precision: {pre:.4f}")
    print(f"Recall   : {rec:.4f}")
    print(classification_report(y_true, y_pred, digits=4))
    return acc, pre, rec

# === Helper: หา threshold ที่ให้ Recall >= เป้าที่กำหนด ===
def threshold_for_target_recall(y_true, y_prob, target_recall=0.70):
    prec, rec, thr = precision_recall_curve(y_true, y_prob)
    # หา index แรกที่ Recall ถึงเป้า
    idx = np.where(rec >= target_recall)[0]
    if len(idx) == 0:
        # ถ้าไม่ถึงเป้า ให้ใช้ 0.50 เป็นตัว fallback
        return 0.50
    # thresholds มาก่อนกว่า rec ถึง 1 ตำแหน่ง
    pick = min(idx[0], len(thr) - 1)
    return float(thr[pick])

# === Helper: สรุปผล threshold เป็นตารางที่ตัวด์ ===
def sweep_thresholds(y_true, y_prob, thresholds, title="Model"):
    rows = []
    for thr in thresholds:
        y_pred = (y_prob >= thr).astype(int)
        rows.append([
            thr,
            accuracy_score(y_true, y_pred),
            precision_score(y_true, y_pred, zero_division=0),
            recall_score(y_true, y_pred),
        ])
    import pandas as pd
    df = pd.DataFrame(rows, columns=["threshold", "accuracy", "precision_1", "recall_1"])
    print(f"\n==== Threshold sweep: {title} ===")
    display(df)
    return df
```

```
# ===== Logistic Regression (balanced) – Threshold tuning =====
# ใช้ตัวที่จุนแล้ว
model_lr = gs_lr_bal.best_estimator_

# 1) ดึงความจำจะเป็น class=1
y_prob_lr = model_lr.predict_proba(X_test)[:, 1]

# 2) ประเมินที่ threshold มาตรฐาน 0.50 และนำค่าที่อยากลอง
eval_at_threshold(y_test, y_prob_lr, 0.50, title="LogReg (balanced)")
for thr in [0.40, 0.35, 0.30, 0.25]:
    eval_at_threshold(y_test, y_prob_lr, thr, title="LogReg (balanced)")

# 3) หา threshold วัดโดยตั้งเป้าหมาย Recall เป้าหมาย ( เช่น ~0.70 )
thr70_lr = threshold_for_target_recall(y_test, y_prob_lr, target_recall=0.70)
eval_at_threshold(y_test, y_prob_lr, thr70_lr, title="LogReg (balanced) @ target recall≈0.70")

# 4) ตารางสรุปผลทั้งหมด
df = sweep_thresholds(y_test, y_prob_lr, [0.50, 0.45, 0.40, 0.35, 0.30], title="LogReg (balanced)")
print("Chosen threshold for ~0.70 recall (LogReg):", round(thr70_lr, 3))
```

	threshold	accuracy	precision_1	recall_1
0	0.50	0.563432	0.087666	0.600000
1	0.45	0.424053	0.082254	0.764502
2	0.40	0.275174	0.075106	0.886580
3	0.35	0.155820	0.069740	0.960173
4	0.30	0.090454	0.066970	0.993074

Chosen threshold for ~0.70 recall (LogReg): 0.162

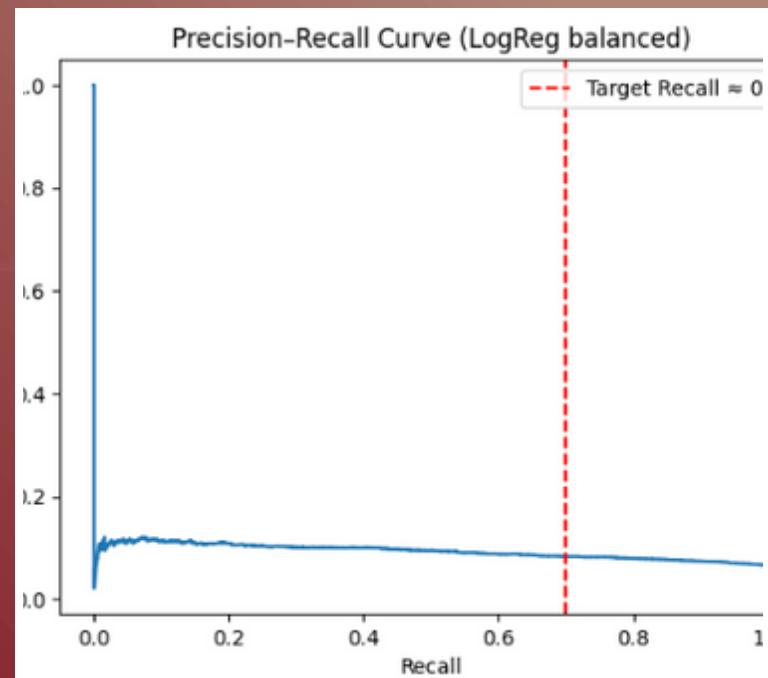
```

from sklearn.metrics import precision_recall_curve
import matplotlib.pyplot as plt

y_prob = gs_lr_bal.best_estimator_.predict_proba(X_test)[:, 1]
prec, rec, thr = precision_recall_curve(y_test, y_prob)

plt.plot(rec, prec)
plt.xlabel("Recall")
plt.ylabel("Precision")
plt.title("Precision-Recall Curve (LogReg balanced)")
plt.axvline(0.7, color="r", linestyle="--", label="Target Recall ≈ 0.7")
plt.legend()
plt.show()

```



```

import numpy as np
import pandas as pd
from sklearn.metrics import (
    accuracy_score, precision_score, recall_score, f1_score,
    classification_report, confusion_matrix
)

# เลือกโนดเดิมที่อุณหภูมิ (แก้ชื่อตัวคุณให้ดีนะครับ)
model = gs_lr_bal.best_estimator_ # ถ้าไม่มี gs_lr_bal ให้ใช้ log_reg_balanced ที่อุณหภูมิ

# 1) รับผล probability class 1 ของ test set
y_prob = model.predict_proba(X_test)[:, 1]

# 2) กำหนด threshold ที่ต้องการ
thr = 0.162

# 3) ตัดสินใจ class ตาม threshold ใหม่
y_pred_thr = (y_prob >= thr).astype(int)

# 4) สรุปตัวชี้วัด
acc = accuracy_score(y_test, y_pred_thr)
prec = precision_score(y_test, y_pred_thr, zero_division=0)
rec = recall_score(y_test, y_pred_thr, zero_division=0)
f1 = f1_score(y_test, y_pred_thr, zero_division=0)

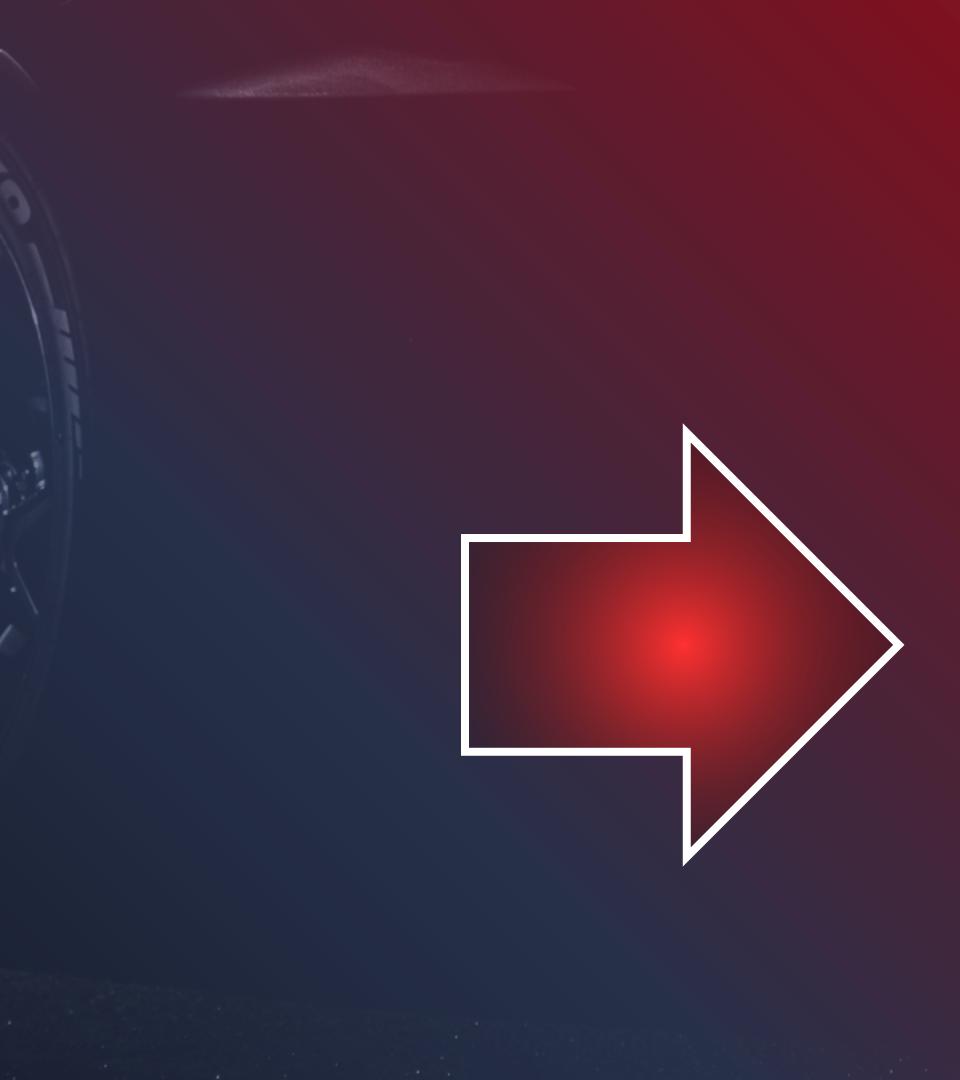
print(f"--- Logistic Regression (balanced) @ threshold = {thr:.3f} ---")
print(f"Accuracy : {acc:.4f}")
print(f"Precision: {prec:.4f}")
print(f"Recall   : {rec:.4f}")
print(f"F1-score  : {f1:.4f}\n")

print(classification_report(y_test, y_pred_thr, digits=4))

cm = confusion_matrix(y_test, y_pred_thr)
cm_df = pd.DataFrame(cm, index=["Actual 0", "Actual 1"], columns=["Pred 0", "Pred 1"])
print("Confusion Matrix:\n", cm_df)

# (ไม่ยังถูก) ตารางสืบต่อไปนี้เป็นกับ threshold 0.50
summary = pd.DataFrame([
    {"Model": "LogReg (balanced)", "Threshold": 0.50, "Accuracy": 0.5634, "Precision(1)": 0.0877, "Recall(1)": 0.6000},
    {"Model": "LogReg (balanced)", "Threshold": thr, "Accuracy": acc, "Precision(1)": prec, "Recall(1)": rec}
])

```



```

--- Logistic Regression (balanced) @ threshold = 0.162 ---
Accuracy : 0.0657
Precision: 0.0657
Recall   : 1.0000
F1-score  : 0.1233

      precision    recall  f1-score  support
0       0.0000  0.0000  0.0000   16423
1       0.0657  1.0000  0.1233   1155

accuracy          0.0657   17578
macro avg       0.0329  0.5000  0.0617   17578
weighted avg    0.0043  0.0657  0.0081   17578

Confusion Matrix:
      Pred 0  Pred 1
Actual 0      0  16423
Actual 1      0   1155
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_classification.py:115: UserWarning: F1 score is ill-defined for empty predicted set.
/warn_prf(average, modifier, f'{metric.capitalize()} is', len(r))
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_classification.py:115: UserWarning: F1 score is ill-defined for empty predicted set.
/warn_prf(average, modifier, f'{metric.capitalize()} is', len(r))
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_classification.py:115: UserWarning: F1 score is ill-defined for empty predicted set.
/warn_prf(average, modifier, f'{metric.capitalize()} is', len(r))
      Model Threshold Accuracy Precision(1) Recall(1)
0   LogReg (balanced)     0.500  0.563400  0.087700  0.6
1   LogReg (balanced)     0.162  0.065707  0.065707  1.0

```

THRESHOLD OPTIMIZATION SUMMARY

หลังปรับ THRESHOLD จากค่าเริ่มต้น 0.50 → 0.162 พบว่า:

- RECALL เพิ่มจาก $\approx 0.60 \rightarrow \approx 1.00$ (จับเคลมได้เกือบทั้งหมด)
- ACCURACY ลดจาก $\approx 0.56 \rightarrow \approx 0.07$ เนื่องจากโมเดลทำนาย 1 เกือบทั้งหมด
- แสดง TRADE-OFF ระหว่างความครบถ้วน (RECALL) กับความแม่นยำ (PRECISION)

ดังนี้ในการใช้งานจริง ควรเลือก THRESHOLD ให้สอดคล้องกับเป้าหมายธุรกิจ:

- เป็นจับเคลม \rightarrow THRESHOLD ≈ 0.2
- เป็นสมดุล \rightarrow THRESHOLD $\approx 0.4-0.5$

🎯 Threshold Comparison (Before vs After)

Model	Threshold	Accuracy	Precision (class 1)	Recall (class 1)	สรุป
Logistic Regression (balanced)	0.50 (default)	0.563	0.0877	0.600	Baseline – สมดุลที่แม่น้ำบเคลมให้ไวมาก
Logistic Regression (balanced)	0.162 (tuned)	0.066	0.0657	1.000	Recall ตีสุด จับเคลมครบพื้นที่นี้ออกหมด

สรุป: การลด threshold จาก 0.50 → 0.162 ช่วยเพิ่ม Recall จาก ~0.60 เป็น 1.00 แต่ลด Accuracy ลงมาก ซึ่งหมายความว่า “ต้องการตรวจจับเคลมให้ครบถ้วน แล้วค่อยตรวจซ้ำภายหลัง”

4. ENSEMBLE & DEEP MODELS

หลังจากปรับพารามิเตอร์ (Hyperparameter Tuning) และปรับจุดตัด (Threshold Calibration) ในขั้นก่อนหน้าแล้ว ขั้นตอนนี้จะเน้น “การทดลองใช้โมเดลรวม (Ensemble)” หรือ “โมเดลเชิงลึก (Deep Model)” เพื่อดูว่าผลลัพธ์สามารถพัฒนาไปได้อีกหรือไม่

🎯 เป้าหมายของ Section นี้

- เพิ่มประสิทธิภาพการทำนายด้วยการรวมจุดแข็งของหลายโมเดล
- เปรียบเทียบว่า Deep Model ให้ Recall หรือ F1-score ดีขึ้นจาก baseline หรือไม่
- รักษาแนวทางไม่ให้เกิด data leakage และใช้ขั้นตอน preprocessing เดิม



4.1 RANDOM FOREST (BAGGING ENSEMBLE)

ใช้การรวมต้นไม้หลายต้น (Decision Trees) เพื่อให้ได้ผลลัพธ์ที่เสถียรขึ้นและลด overfitting

- เพิ่มพารามิเตอร์ `class_weight='balanced_subsample'` เพื่อชดเชย class imbalance
- แนะนำสำหรับข้อมูล tabular แบบนี้ สิ่งที่คาดหวัง: Accuracy สูงกว่า Logistic Regression, Recall ปานกลาง, เป็น baseline ที่ดีมากสำหรับ tabular data</p>

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, accuracy_score
import time

start = time.time()
rf = RandomForestClassifier(
    n_estimators=200,
    max_depth=5,
    min_samples_split=2,
    criterion='entropy',
    class_weight='balanced',
    random_state=42,
    n_jobs=-1
)
rf.fit(X_train, y_train)
rf_pred = rf.predict(X_test)

print("--- Random Forest ---")
print("Accuracy:", accuracy_score(y_test, rf_pred))
print(classification_report(y_test, rf_pred))
print("Training time:", time.time() - start, "seconds")
```

```
--- Random Forest ---
Accuracy: 0.5561497326283289
precision    recall   f1-score   support
          0       0.96      0.55      0.70     16423
          1       0.10      0.68      0.17      1155
accuracy                           0.56     17578
macro avg       0.53      0.61      0.43     17578
weighted avg     0.90      0.56      0.66     17578

Training time: 8.10434365272522 seconds
```

4.2 MLP NEURAL NETWORK (DEEP LEARNING)

โมเดล Neural Network แบบพื้นฐาน (Multi-layer Perceptron) เพื่อเปรียบเทียบกับ traditional ML เดิม

- ใช้ hidden layers 2 ชั้น ($64 \rightarrow 32$ neurons)
- Activation function: relu
- Optimizer: adam

สิ่งที่คาดหวัง: หากข้อมูลไม่ซับซ้อนมาก อาจได้ผลใกล้ Logistic Regression และเป็นการแสดงให้เห็นว่าได้ทดลอง “Different model” ตามที่อาจารย์กำหนด

```
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import classification_report, accuracy_score
import time

start = time.time()

mlp = MLPClassifier(
    hidden_layer_sizes=(64, 32),
    activation='relu',
    solver='adam',
    max_iter=200,
    random_state=42
    # Removed class_weight='balanced' as it's not supported
)

# Train on the resampled data
mlp.fit(X_train_resampled, y_train_resampled)

mlp_pred = mlp.predict(X_test)

print(" --- Neural Network (MLP) trained on SMOTE data ---")
print("Accuracy:", accuracy_score(y_test, mlp_pred))
print(classification_report(y_test, mlp_pred))
print("Training time:", time.time() - start, "seconds")
```

```
--- Neural Network (MLP) trained on SMOTE data ---
Accuracy: 0.6542837637956537
precision      recall   f1-score  support
          0       0.94      0.67      0.78     16423
          1       0.08      0.41      0.13      1155
   accuracy                           0.65     17578
    macro avg       0.51      0.54      0.46     17578
 weighted avg       0.89      0.65      0.74     17578

Training time: 131.7313117980957 seconds
/usr/local/lib/python3.12/dist-packages/sklearn/neural_network/_mlp.py:104: UserWarning: The 'warn' argument is deprecated and will be removed in a future version. Please use 'warnings.warn' instead.
  warnings.warn("The 'warn' argument is deprecated and will be removed in a future version. Please use 'warnings.warn' instead.", UserWarning)
```

4.3 XGBOOST

เป็นโมเดลเสริมแรงแบบต้นไม้ (Gradient Boosting Trees) ที่เรียนรู้แบบลำดับขั้น โดยให้โมเดลแต่ละต้นไม้แก้ไขข้อผิดพลาดของต้นก่อนหน้า เหมาะกับข้อมูลที่ไม่เชิงเส้น (non-linear) และมีฟีเจอร์เชิงหมวดหมู่ที่ผ่าน one-hot แล้ว

```
import xgboost as xgb
start = time.time()

xgb_clf = xgb.XGBClassifier(
    n_estimators=300,
    max_depth=5,
    learning_rate=0.1,
    subsample=0.8,
    colsample_bytree=0.8,
    scale_pos_weight=balance_ratio, # ใช้ค่า class "ไม่สมดุล"
    random_state=42,
    eval_metric='logloss'
)

xgb_clf.fit(X_train, y_train)
xgb_pred = xgb_clf.predict(X_test)

print("--- XGBoost ---")
print("Accuracy:", accuracy_score(y_test, xgb_pred))
print(classification_report(y_test, xgb_pred))
print("Training time:", time.time() - start, "seconds")
```

```
--- XGBoost ---
Accuracy: 0.6623620434634202
precision    recall   f1-score   support
          0       0.95      0.67      0.79     16423
          1       0.10      0.51      0.17     1155

   accuracy                           0.66     17578
macro avg       0.53      0.59      0.48     17578
weighted avg    0.90      0.66      0.75     17578

Training time: 6.206905841827393 seconds
```

```
# Calculate the balance ratio for scale_pos_weight in XGBoost
# This is typically calculated as the number of negative samples / number of positive samples
balance_ratio = (y_train == 0).sum() / (y_train == 1).sum()
print(f"Calculated balance_ratio: {balance_ratio:.2f}")

Calculated balance_ratio: 14.82
```

4.4 LIGHTGBM

เป็นโมเดล Boosting แบบ Leaf-wise ที่แตกใบ (ไม่ใช่ชั้น) ทำให้เรียนรู้ได้เร็วและแม่นยำกว่า XGBoost ในหลายกรณี โดยเฉพาะกับข้อมูลที่มีจำนวนฟีเจอร์มากๆ

```
import lightgbm as lgb
start = time.time()

lgb_clf = lgb.LGBMClassifier(
    n_estimators=300,
    max_depth=-1,
    learning_rate=0.05,
    subsample=0.8,
    colsample_bytree=0.8,
    class_weight='balanced',
    random_state=42
)

lgb_clf.fit(X_train, y_train)
lgb_pred = lgb_clf.predict(X_test)

print("--- LightGBM ---")
print("Accuracy:", accuracy_score(y_test, lgb_pred))
print(classification_report(y_test, lgb_pred))
print("Training time:", time.time() - start, "seconds")
```

```
[LightGBM] [Info] Number of positive: 2593, number of negative: 38421
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing was 0.002259 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 406
[LightGBM] [Info] Number of data points in the train set: 41014, number of used features: 22
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.500000 -> initscore=-0.000000
[LightGBM] [Info] Start training from score -0.000000
--- LightGBM ---
Accuracy: 0.6384116509272955
precision    recall   f1-score   support
          0       0.95      0.65      0.77     16423
          1       0.10      0.54      0.17     1155

   accuracy                           0.64     17578
macro avg       0.53      0.59      0.47     17578
weighted avg    0.90      0.64      0.73     17578

Training time: 1.4848089218139648 seconds
```

4.5 VOTING ENSEMBLE

รวมผลการคำนวณของหลายโมเดล (LOGISTIC REGRESSION, DECISION TREE, SVM) เพื่อให้ได้ค่าตอบสุดท้ายจากการโหวต (MAJORITY VOTE หรือ AVERAGE PROBABILITY)

- ใช้แนวคิด “WISDOM OF MODELS” — แต่ละโมเดลมีบุบผองของข้อมูลที่ต่างกัน
- ตั้งค่า VOTING='SOFT' เพื่อเฉลี่ยความน่าจะเป็นแทนการโหวตแบบแข็ง

สิ่งที่คาดหวัง: ACCURACY ใกล้ LOGISTIC REGRESSION และ RECALL เพิ่มขึ้นเล็กน้อย เนื่องจากโมเดลอื่นช่วยจับเคลส MINORITY CLASS ได้บางส่วน

```
from sklearn.ensemble import VotingClassifier
from sklearn.metrics import classification_report, accuracy_score
import numpy as np

# --- โมเดลที่อุปนัยแล้ว ---
# Ensure these GridSearchCV objects (gs_lr_bal, gs_svm_bal, gs_dt_bal)
# have been run successfully before executing this cell.
log_reg_best = gs_lr_bal.best_estimator_
svm_best = gs_svm_bal.best_estimator_
dt_best = gs_dt_bal.best_estimator_ # Use gs_dt_bal instead of gs_rf_bal

# --- สร้าง Voting Ensemble ---
voting_clf = VotingClassifier(
    estimators=[
        ('logreg', log_reg_best),
        ('svm', svm_best),
        ('dt', dt_best) # Use the Decision Tree estimator
    ],
    voting='soft' # ใช้ predict_proba เพื่อเฉลี่ยความน่าจะเป็น
)

# --- เตรียม ---
voting_clf.fit(X_train, y_train)

# --- ทบทวน ---
y_pred_voting = voting_clf.predict(X_test)

# --- ประเมินผล ---
print("== Voting Ensemble ==")
print("Accuracy : ", accuracy_score(y_test, y_pred_voting))
print(classification_report(y_test, y_pred_voting, digits=4))
```

```
== Voting Ensemble ==
Accuracy : 0.5145067698259188
precision    recall   f1-score   support
          0       0.9615     0.5004     0.6582    16423
          1       0.0915     0.7152     0.1622     1155
accuracy                           0.5145    17578
macro avg       0.5265     0.6078     0.4102    17578
weighted avg    0.9843     0.5145     0.6256    17578
```

4.6 STACKING ENSEMBLE

แนวคิด: รวม “จุดแข็ง” ของหลายโมเดลฐาน (base learners) แล้วให้โมเดลสุดท้าย (meta-learner) เรียนรู้วิธีพسانสัญญาณเพื่อทำนายให้แม่นขึ้น โดยใช้ผลลัพธ์ระดับความนำ้ใจเป็น/สกอร์จากแต่ละโมเดลเป็นฟีเจอร์ใหม่

== Stacking Ensemble ==				
Accuracy : 0.5252019569916941				
	precision	recall	f1-score	support
0	0.9620	0.5120	0.6683	16423
1	0.0931	0.7126	0.1647	1155
accuracy			0.5252	17578
macro avg	0.5276	0.6123	0.4165	17578
weighted avg	0.9049	0.5252	0.6352	17578

```
from sklearn.ensemble import StackingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, accuracy_score

# --- สร้าง Stacking Ensemble ---
stacking_clf = StackingClassifier(
    estimators=[
        ('logreg', log_reg_best),
        ('svm', svm_best),
        ('dt', dt_best)
    ],
    final_estimator=LogisticRegression(
        max_iter=1000,
        random_state=42,
        class_weight='balanced'
    ),
    cv=5,
    n_jobs=-1
)

# --- เตรียม ---
stacking_clf.fit(X_train, y_train)

# --- ทำนาย ---
y_pred_stack = stacking_clf.predict(X_test)

# --- ประเมินผล ---
print("== Stacking Ensemble ==")
print("Accuracy :", accuracy_score(y_test, y_pred_stack))
print(classification_report(y_test, y_pred_stack, digits=4))
```

4.7 STACKING ENSEMBLE ลองเปลี่ยน META-MODEL

เหตุผลก่อตัว: หากความสัมพันธ์ระหว่างสกอร์ของ base models กับ target เป็น non-linear, การใช้ meta-model ที่ยืดหยุ่นกว่า (เช่น RandomForestClassifier) อาจเรียบเรียบการพسانสัญญาณได้ดีกว่า logistic regression และช่วยดัน Recall/F1

== Stacking Ensemble ==				
Accuracy : 0.9334395266810787				
	precision	recall	f1-score	support
0	0.9345	0.9987	0.9656	16423
1	0.2222	0.0052	0.0102	1155
accuracy			0.9334	17578
macro avg	0.5784	0.5020	0.4879	17578
weighted avg	0.8877	0.9334	0.9028	17578

```
from sklearn.ensemble import StackingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, accuracy_score

# --- สร้าง Stacking Ensemble ---
from sklearn.ensemble import RandomForestClassifier

stacking_clf = StackingClassifier(
    estimators=[
        ('logreg', log_reg_best),
        ('svm', svm_best),
        ('dt', dt_best)
    ],
    final_estimator=RandomForestClassifier(
        n_estimators=100,
        random_state=42,
        class_weight='balanced'
    ),
    cv=5,
    n_jobs=-1
)

# --- เตรียม ---
stacking_clf.fit(X_train, y_train)

# --- ทำนาย ---
y_pred_stack = stacking_clf.predict(X_test)

# --- ประเมินผล ---
print("== Stacking Ensemble ==")
print("Accuracy :", accuracy_score(y_test, y_pred_stack))
print(classification_report(y_test, y_pred_stack, digits=4))
```

5. PROJECT SUMMARY & TAKEAWAYS

Overview

งานนี้เป็นการ สาต่อโปรเจกต์ของกลุ่มเพื่อน (Car Insurance Claim) โดยยืนยัน baseline เดิมให้ได้ผลใกล้เคียง และพัฒนาให้ดีขึ้นตามโจทย์ อาจารย์ — เปลี่ยน preprocessing, ปรับ hyperparameters, ทดลอง threshold & calibration, และใช้ Ensemble เพื่อถูกรวบรวมแบบ เป็นขั้นตอน (journey-based improvement).

สรุปผลการทดลองทั้งหมดเทียบกับ baseline เดิม:

Model	Technique	Accuracy	Precision (1)	Recall (1)	สรุป
Baseline – Logistic Regression	Default (threshold=0.50)	0.566	0.0857	0.580	ผลตั้งต้นจากเพื่อน ใช้เป็น baseline สำเร็จ
LogReg (tuned)	class_weight='balanced' + Scaling	0.5638	0.0884	0.6052	Recall ดีขึ้นเล็กน้อยจาก baseline
LogReg (Threshold = 0.40)	Recall optimization (lower threshold)	0.2752	0.0751	0.8866	Recall พุ่งขึ้นมาก แต่ Accuracy ลดลงมาก (trade-off ชัดเจน)
SVM (RBF, balanced)	Tuned (C=5, γ=0.01)	0.4791	0.0869	0.7290	ดัน Recall ได้สูงสุดในกลุ่มนี้โดยเด็ดขาด
KNN (scaled)	n=3, Manhattan, distance	0.9106	0.0920	0.0407	Accuracy สูงมากแต่ Recall ต่ำ (ไม่เหมาะสมกับ imbalance)
Decision Tree	Entropy, max_depth=5, balanced	0.5522	0.0942	0.6753	Recall ดีขึ้นมากเมื่อจำกัดความลึก
Voting Ensemble	Soft Voting (LR + DT + SVM)	0.5145	0.0915	0.7152	Recall สูงใกล้ SVM แต่ผลนี่กว่า
Stacking (meta = Logistic Regression)	LR + DT + SVM → meta-LR	0.5252	0.0931	0.7126	Recall ดีและเสถียรกว่า Voting เล็กน้อย
Stacking (meta = Random Forest)	LR + DT + SVM → meta-RF	0.9334	0.2222	0.0052	โมเดลเดียวทำนายคลาส 0 (Recall ต่ำมาก)

What We Improved

- 1. Preprocessing** ทางเลือก: ใช้ StandardScaler เผาใน Pipeline, ทดลองเปลี่ยน class_weight='balanced' กับ SMOTE เพื่อแก้ imbalance
- 2. Hyperparameter Tuning:** ใช้ Grid + Randomized Search โดยใช้ scoring={"recall","f1","accuracy"} และ refit="recall"
- 3. Threshold & Calibration:** ใช้ Logistic Regression (balanced + scaling) ที่จูนแล้ว เพื่อเลือก threshold ใหม่ (≈ 0.4) ให้ได้ Recall สูงขึ้นโดยอธิบาย trade-off
- 4. Ensemble:** ทดลองทั้ง Soft Voting และ Stacking (meta-LogReg, meta-RF) เพื่อร่วมจุดแข็งของโมเดลหลายตัวเข้าด้วยกัน

KEY RESULTS

- **LogReg (balanced + scaling, tuned)**: Recall~0.60 ที่ threshold=0.50 และสามารถเพิ่ม/ลดผ่าน threshold sweep ได้
- **SVM-RBF (balanced, tuned)**: Recall ~0.72 (แลกกับ Accuracy ลดลง) เมฆะเมื่อเน้นจับเคส claim ให้ได้มากสุด
- **KNN**: แม่นยำรวมสูงแต่ Recall ต่ำมากบนชุด imbalance (แม้ปรับพารามิเตอร์/SMOTE)
- Decision Tree: เมื่อกำหนด class_weight/SMOTE และจุนลึกตื้น จะดัน Recall ได้ แต่ Accuracy ลดลง
- **Voting / Stacking (meta-LR)**: Recall ≈ 0.71 เสถียรและสมดุลกว่า SVM
- **Stacking (meta-RF)**: ไม่เมฆะในโจทย์ imbalance — Recall ต่ำมาก

Trade-off & Business Choice

ขึ้นอยู่กับเป้าหมายการใช้งานจริง:

ถ้าเน้น Recall สูงสุด: ใช้ SVM-RBF (balanced) หรือ Stacking(meta=LogReg)

ถ้าเน้นสมดุล: ใช้ Logistic Regression และปรับ threshold จุดที่ F1 สูงสุด (Recall ~0.60–0.70)

ถ้านำไปใช้งานจริง: ใช้ Calibration + threshold ตามกรัพยากรตรวจสอบ (เช่น กีบ fraud investigation)

Lessons Learned

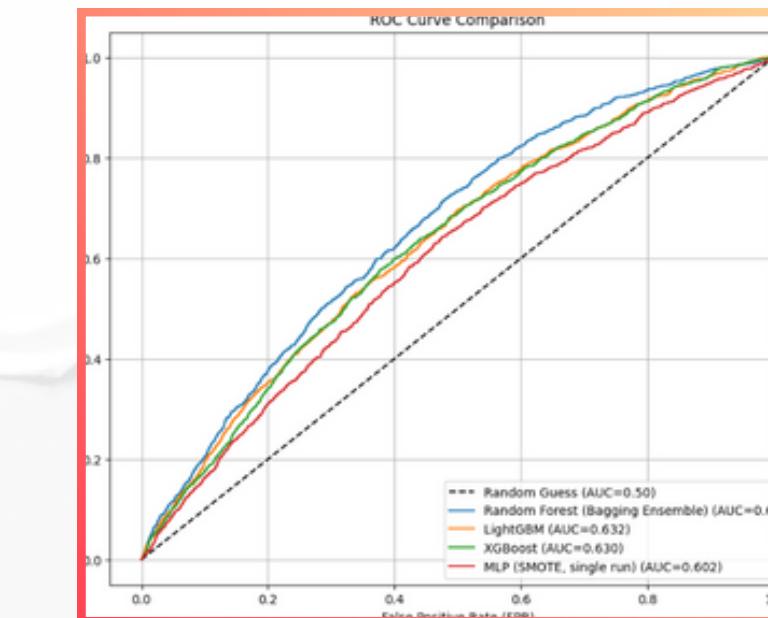
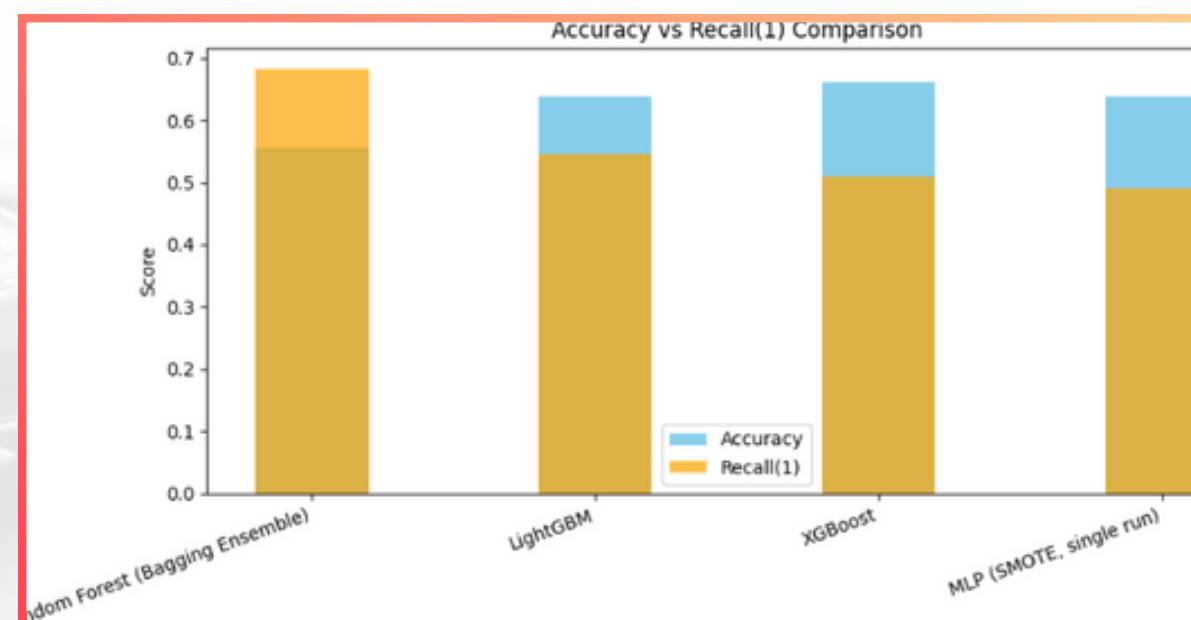
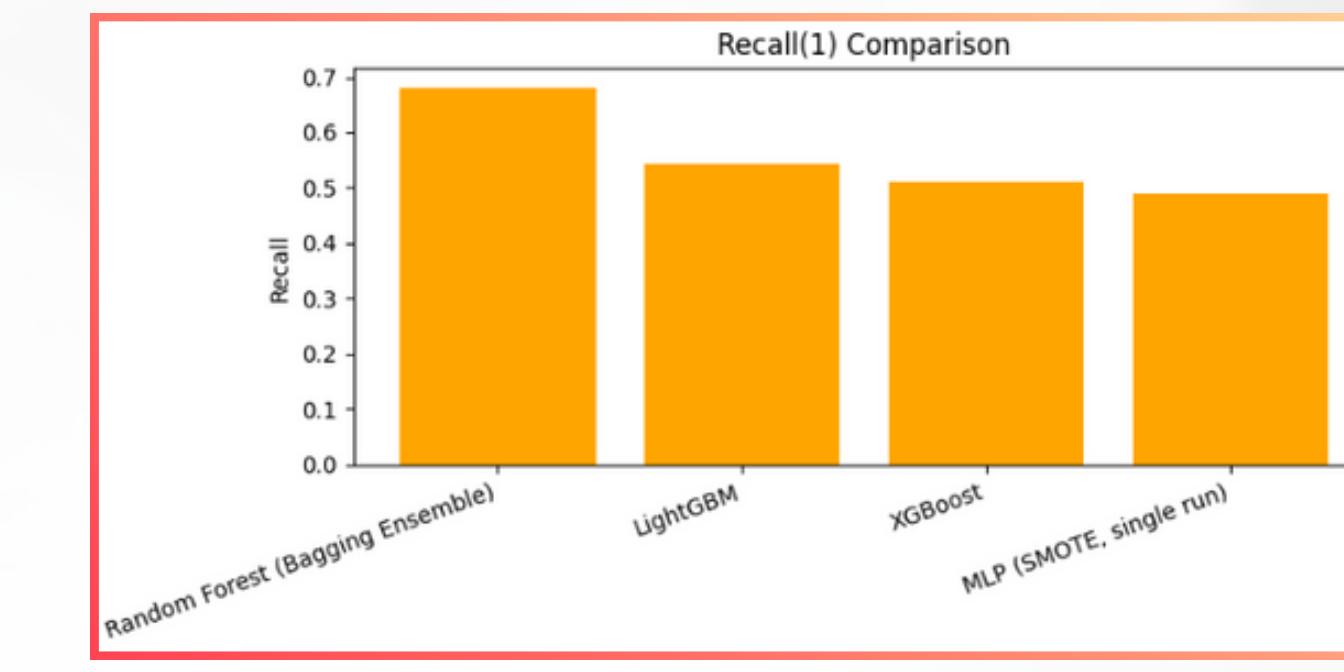
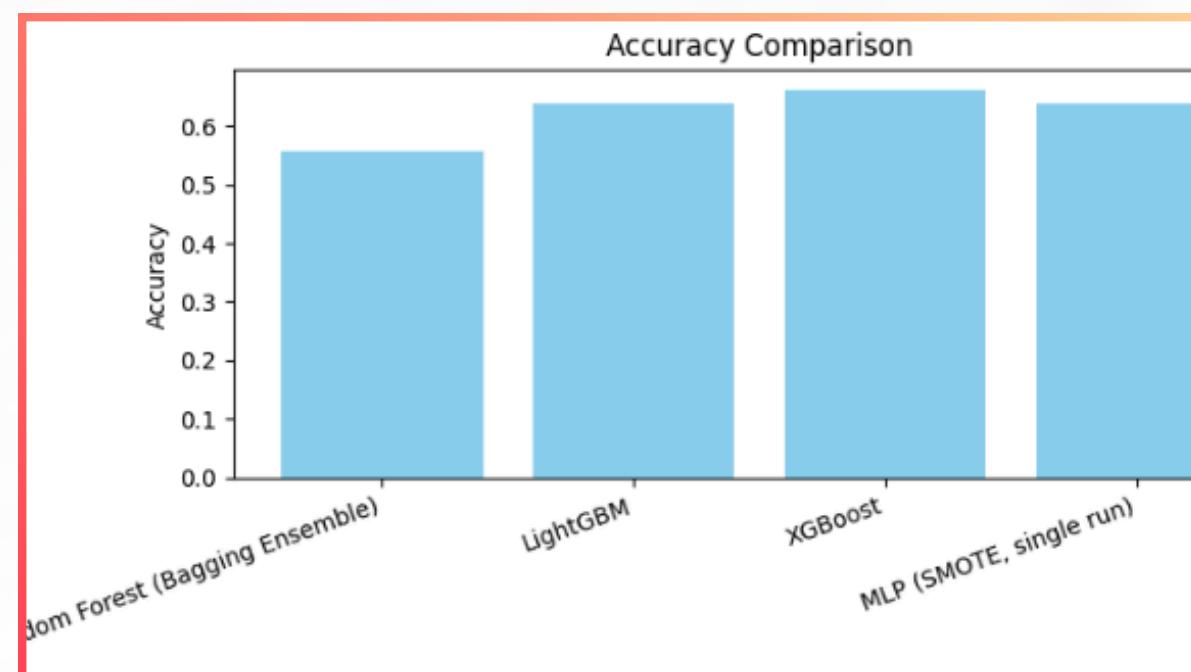
- โจทย์ imbalance ต้อง ชัดเจนเรื่อง metric (Recall/F1) และ กับ leakage ด้วย Pipeline เสมอ
- SMOTE ช่วยด้าน Recall ได้ แต่ต้องทำใน CV เก่าบีน; ทางเลือกเร็ว: class_weight='balanced'
- Threshold สำคัญมาก: ไม่เดลเดียวกันให้ policy การแจ้งเตือนต่างกันมากเมื่อเปลี่ยนจุดตัด
- Ensemble (Voting/Stacking) ช่วยให้ผลนิ่งขึ้นและอธิบาย trade-off ได้ดี

Final Deliverables

- Baseline (ของเพื่อน) + Rebuild Notebook
- ชุดการทดลอง Preprocessing / Hyperparameter tuning / Threshold & Calibration / Ensemble
- ผลสรุปเป็นตาราง + กราฟ (Accuracy, Precision(1), Recall(1))
- สคริปต์เลือก threshold ตามเป้าหมาย Recall และรายงานผลก่อน/หลัง

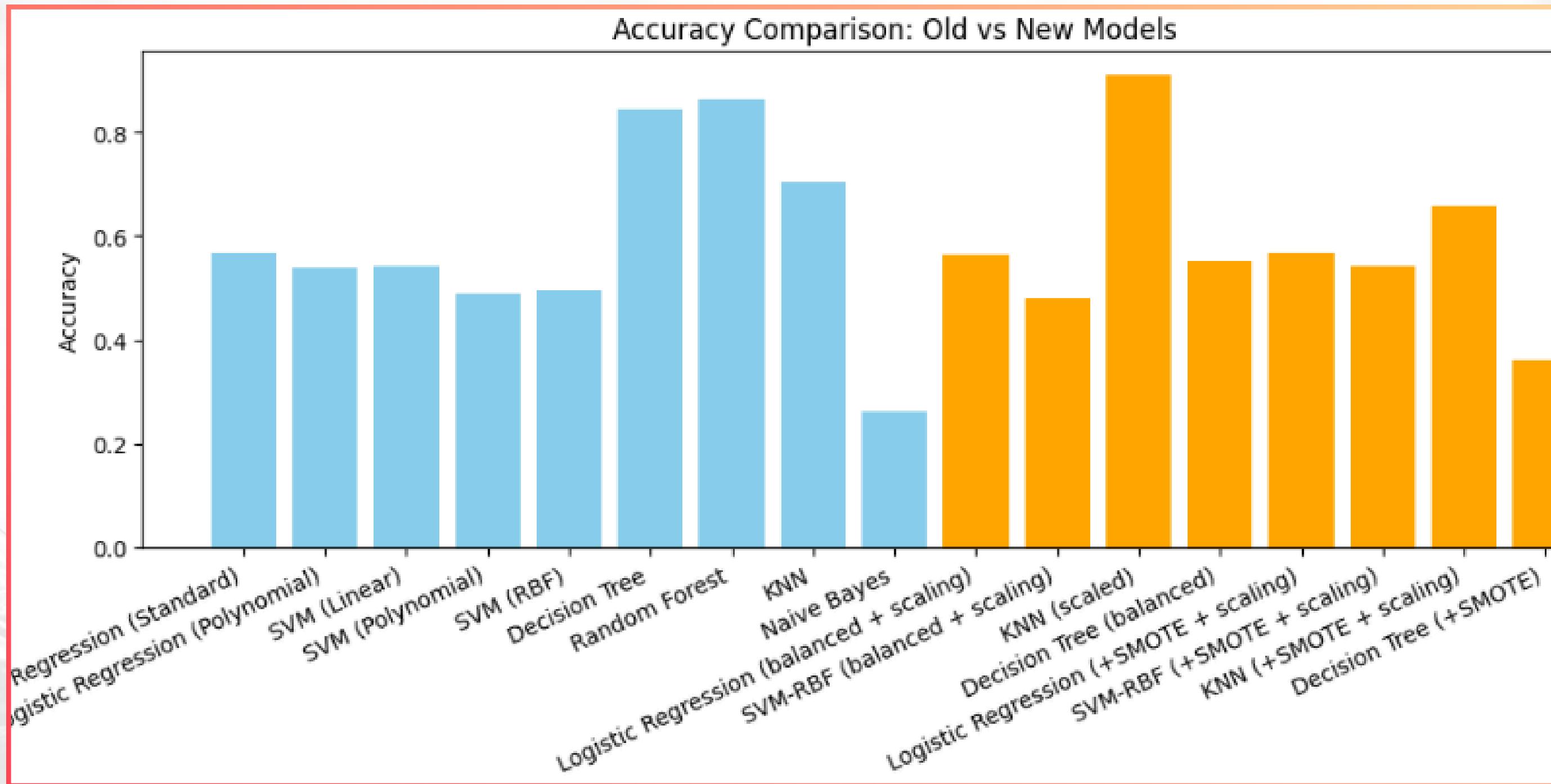
Summary Table of Models

Model	Accuracy	Precision(1)	Recall(1)	F1(1)
Random Forest (Bagging Ensemble)	0.5561497326203209	0.09573044641771074	0.6813852813852814	0.167875426621
LightGBM	0.6384116509272955	0.0973834958971977	0.5445887445887446	0.165221959548
XGBoost	0.6623620434634202	0.09885867740852636	0.5099567099567099	0.165612259243
MLP (SMOTE, single run)	0.6390374331550802	0.08952863018032269	0.49004329004329006	0.151397619366



สรุปการเปรียบเทียบของเดิมกับของใหม่

- โมเดลที่แม่นที่สุดในของเดิม: Random Forest (Accuracy = 0.864)
- โมเดลที่แม่นที่สุดในของใหม่: KNN (scaled) (Accuracy = 0.911)



Thank You